

A decorative graphic on the left side of the slide consisting of two overlapping parallelograms. The front one is blue and the back one is light green. Both are tilted at an angle.

Protocol Buffer

Benyamin Mousavi



چه خواهیم آموخت؟

1. آشنایی با Protocol buffer
2. آشنایی با سرویس و تعریف آن
3. آشنایی با پیام و تعریف آن

1. با تاریخچه آشنا شدیم
2. نیاز به شروع توسعه
3. گام اول آشنایی با Protocol Buffer



معرفی Protocol Buffer

1. یک زبان بسیار ساده و قواعد بسیار ساده ای دارد که ما میتوانیم service ها و message هایی که داخل سرویس های gRPC رد و بدل میشوند را پیاده سازی کنیم.

2. عدم پشتیبانی پیام ها از ارث بری



سه بخش اساسی

1. تعریف کلی مثل نسخه protocol buffer و ...
2. تعریف سرویس
3. تعریف پیام

تعاریف اولیه

1. تعریف نسخه اجباری

2. proto1 , proto2 , proto3

3. نسخه مورد استفاده 3

```
person.proto > ...  
1   syntax = "proto3";  
2  
3   package PERSON;  
4
```



ویژگی های مهم نسخه 3

1. JSON Encoding
2. Well-Known Type
3. Strict UTF-8 Enforcement



آشنایی با تعاریف

1. تعریف سینتکس با `syntax = "proto3"`

2. فضای نام `package mytest.v2`

3. امکان تعیین فضای نام با `option csharp_namespace`

4. استفاده از نام پکیج برای نام سرویس

تعریف سرویس

1. بخش دوم تعریف سرویس

2. تعریف سرویس , توابع و پارامترها

3. تعریف با service

4. شروع تعریف توابع با rpc

```
1 syntax = "proto3";
2
3 package pb;
4 option go_package="./pb";
5
6 message Greeting {
7     string first_name = 1;
8     string last_name = 2;
9 }
10
11 message GreetRequest {
12     Greeting greeting = 1;
13 }
14
15 message GreetResponse {
16     string result = 1;
17 }
18
19 service GreetService{
20     rpc Greet(GreetRequest) returns (GreetResponse) {};
21 }
```



5. نیاز به باز نویسی توابع

6. RpcException پیش فرض

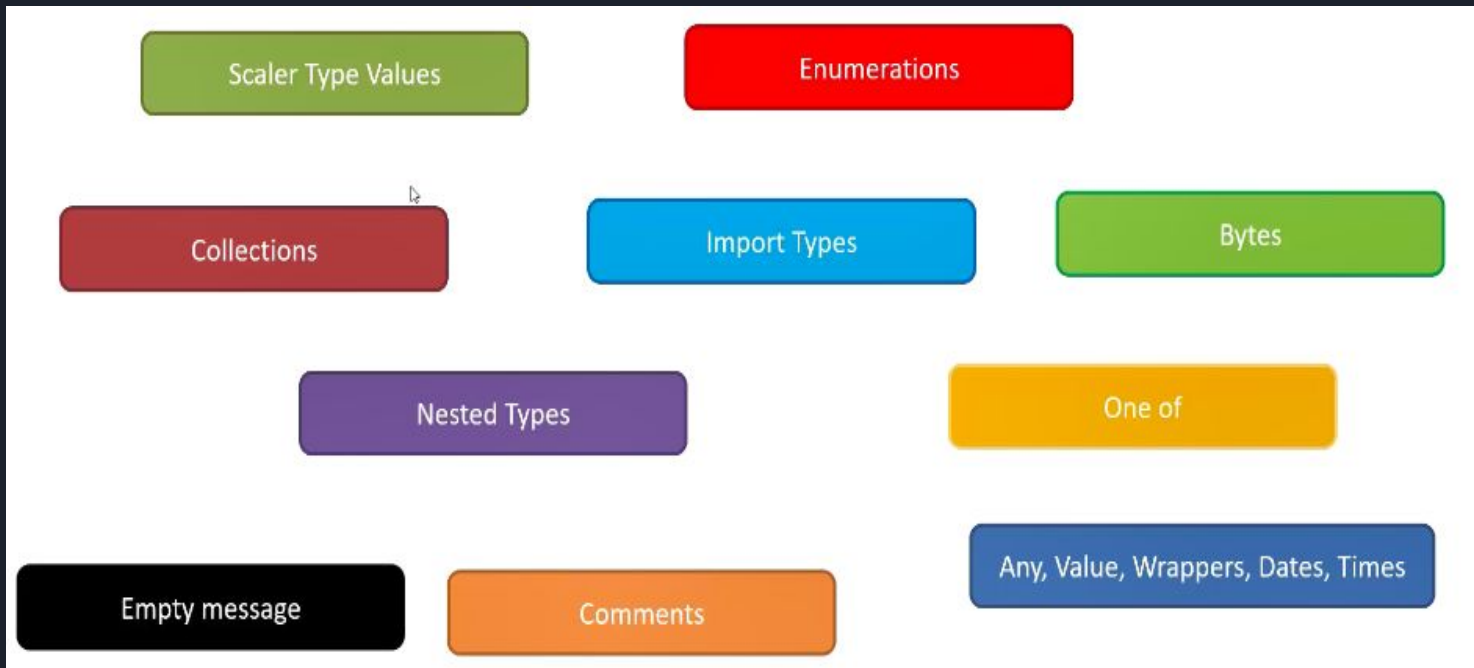
5. عدم استفاده از Http Verbs

6. نیاز به راهکاری جهت تعیین نوع

7. GetKeyword , DeleteKeyword

8. DeleteRequest , DeleteReply

تعريف پیام




ویژگی های پیام و سرویس

```
1 syntax = "proto3";
2
3 package PERSON;
4
5 message Person {
6     optional string name = 1;
7     optional int32 id = 2;
8     optional string email = 3;
9 }
10
11 message AddressBook {
12     repeated Person people = 1;
13 }
```

1. عدم پشتیبانی از ارث بری
2. عدم پشتیبانی از صحت سنجی
3. عدم امکان تعریف بدون ورودی و خروجی
4. تعریف پیام با کلمه کلیدی message
5. ویژگی های پیش فرض Optional
6. اجبار به تعریف موقعیت برای Serialization

انواع Scaler

Default Value	C# Type	Proto buffer type
0	double	double
0	float	float
0	int	int32
0	long	int64
0	uint	uint32
0	ulong	uint64
0	int	sint32
0	long	sint64



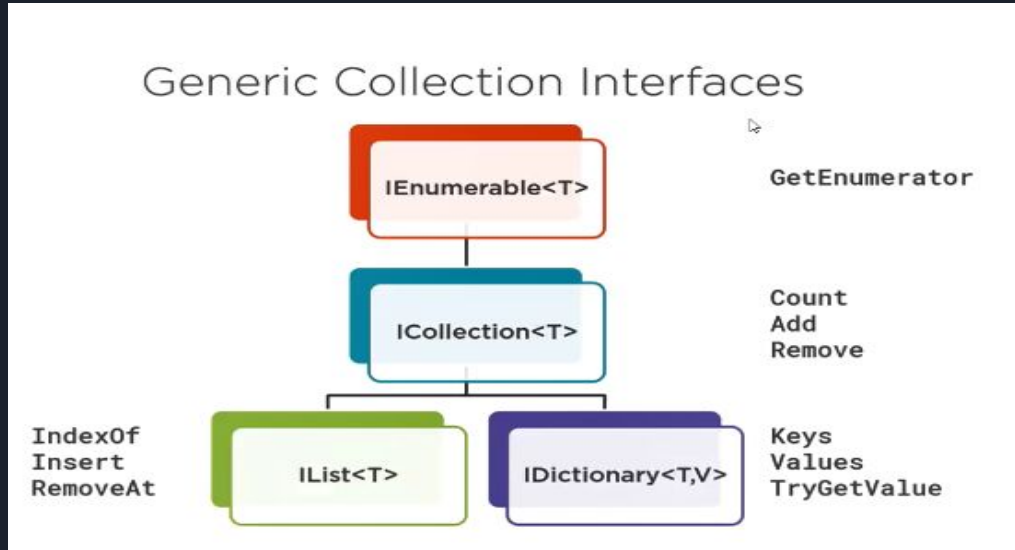
Default Value	C# Type	Proto buffer type
0	uint	fixed32
0	ulong	fixed64
0	int	sfixed32
0	long	sfixed64
false	bool	bool
String.Empty	string	string
Empty bytes	ByteString	byte

مجموعه ها در Protocol Buffer

1. یکی از انواع بسیار مهم

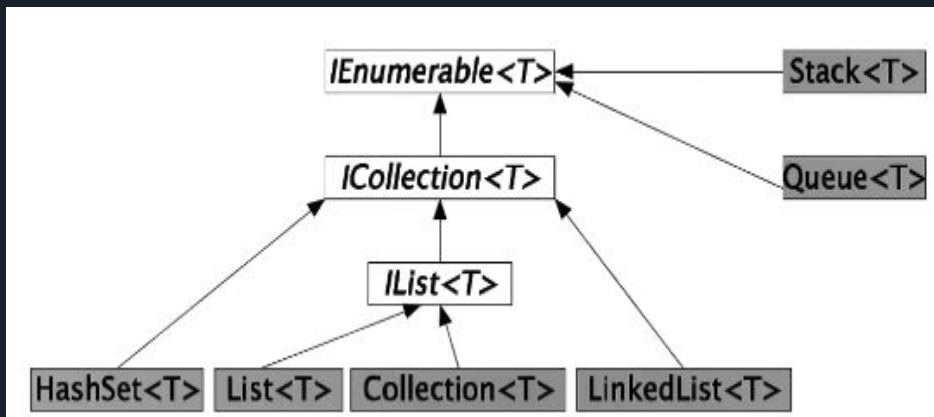
2. Lists

3. Dictionaries

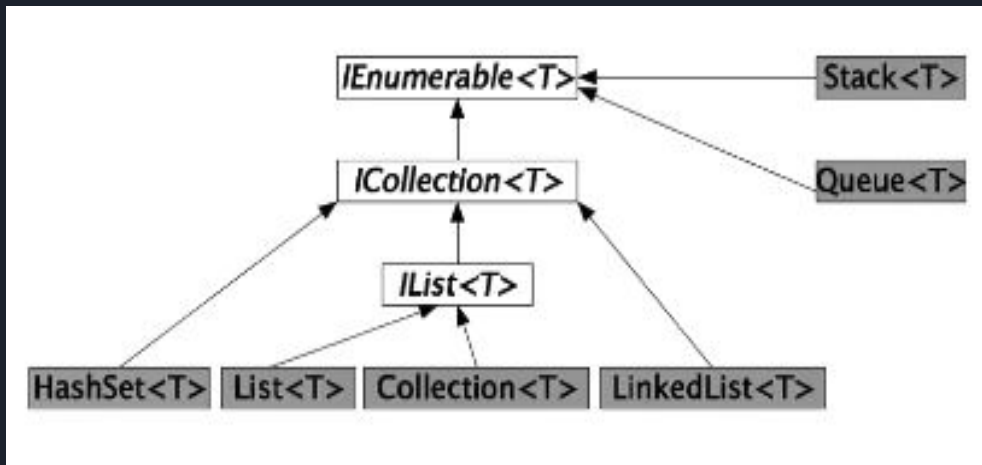


List

1. تعریف با repeated
2. repeated int32 TagIds=1
3. تبدیل به `<RepeatedField<T>`
4. یک کلاس Sealed
5. پیاده سازی بسیاری از مجموعه های NET.



Dictionary



1. `map<key,value >`
2. `; map <int32 , string> students = 1`
3. تبدیل به `MapField`
4. پیاده سازی `IDictionary`

Enum در Proto buffer

```
1  syntax = "proto3";
2
3  enum MessageType {
4      MESSAGE_TYPE_INVALID = 0;
5      MESSAGE_TYPE_UNSET = 1;
6      MESSAGE_TYPE_EMERGENCY = 2;
7      MESSAGE_TYPE_UPDATE = 3;
8  }
9  // My First Message in Protocol Buffers
10 message FirstMessage{
11     int32 id = 1;
12     string msg = 2;
13     string sender = 3;
14     MessageType msg_type = 4;
15 }
```

1. کاملاً شبیه به C#
2. اجبار به تعریف مقدار هر کلید
3. امکان استفاده به عنوان Property
4. امکان تعریف aliases
5. فعال سازی با `allow_alias = true`



انواع تو در تو Nested Type

1. می توان یک نوع را داخل نوعی دیگر تعریف کرد
2. نام نوع و ویژگی باید متفاوت باشد



Import Type

1. امکان تعریف انواع در چندین فایل
2. در صورت نیاز به یک نوع `Import`
3. استفاده از کلمه `import`
4. تعیین محل برای وارد کردن



آشنایی با Well-known Types

Any

Value

Dates

Wrappers

Structs

Times



Any

1. Strongly Type نیست
2. امکان ارسال هر نوع داده
3. `import "google/protobuf/any.proto"`
4. `google.protobuf.Any MyAnyType = 2 ;`
5. مقدار دهی با `Any.Pack`
6. دریافت با `Unpack` یا `TryUnpack`
7. چک کردن با متد `Is`
8. بازگشت `null` در صورت عدم موفقیت `unpack`

17. The Request Message Containing the User's Name:

```
message HelloRequest {  
    string name = 1;  
    bool isRequired = 2;  
    repeated int32 age = 3;  
    map<int32, string> students = 4;  
    google.protobuf.Any MyAnyType = 5;  
}
```

```
public override Task<HelloReply> SayHello(HelloRequest request, ServerCallContext context)  
{  
    HelloReply hr = new HelloReply();  
  
    request.MyAnyType = Any.Pack(hr);  
}
```



Wrappers

1. عدم پشتیبانی از Nullable
2. پشتیبانی از Nullable به کمک Wrappers
3. google.protobuf.BoolValue
4. google.protobuf.DoubleValue
5. google.protobuf.FloatValue
6. google.protobuf.Int32Value
7. google.protobuf.Int64Value
8. google.protobuf.UInt32Value
9. google.protobuf.UInt64Value
10. google.protobuf.StringValue
11. google.protobuf.BytesValue
12. Import "google/protobuf/wrappers.proto"



Value

1. کاربرد برای انواع ناشناخته
2. استفاده از `Value.For ...`
3. خواندن با `NumberValue` و ...

مثل داینامیک میباشد.



تاریخ و زمان

1. مقادیر NET. معادلی ندارند

2. "google/protobuf/duration.proto"

3. ارائه TimeSpan

4. google.protobuf.Duration

5. "google/protobuf/timestamp"

6. ارائه DateTime/DateTimeOffset

7. google.protobuf.Timestamp

Duration.FromTimeSpan .1

Timestamp.FromDateTime .2

ToTimeSpan(); .3

ToDateTime(); .4



Bytes

1. امکان انتقال مقادیر Binary

2. راهکاری برای انتقال فایل

3. خواندن با

`ByteString.CopyForm` , `ByteString.FormStream`

4. نوشتن با `WriteTo`



One of

1. انتخاب یکی از چند مقدار

2. در پشت صحنه switch case

3. امکان بازگشت یکی از چند نوع پاسخ

نکته :

خیلی وقت ها چندین جواب مختلف داریم که از متمدون برمیگرده ولی یکی از ان برمیگردد

یعنی مثلا من میتونم یک متد را صدا بزنم برای ادد کردن اگر موفقیت آمیز بودن

پیام بدیم که اون انتیتی که ادد شده را برگردانیم

اگر ناموفق بود باید خطا برگردونم

پس ما دو response مختلف داریم

یکی Response صحیح و یکی Response ناصحیح

و این که محدود به یکی دوتا نیستیم

هر چند تا میتونیم داشته باشیم مثلا یک تایپی میگیریم با توجه به اون تایپ ورودی میخوایم Response type مون متفاوت باشد

در one of چندین پراپرتی تعریف میکنیم

ولی همیشه فقط و فقط یکی از ان مقدار میگیره و مقدارش برگشت داده میشه

این کمک میکنه استفاده بهتری از حافظه داشته باشیم

استفاده بهتری از زیر ساختی که داریم بکنیم و اطلاعات اضافی را جابجا نکنیم

یعنی وقتی کارمون با خطا مواجه شده لزومی نداره که payload اطلاعات خراب را داشته باشیم

از one of استفاده میکنیم که یکی از دیتا تایپ ها را برگردانیم



Empty Message

1. تعیین پارامتر اجباری است

2. امکان استفاده از Empty

```
import "google/protobuf/Empty.proto";.3
```

```
google.protobuf.Empty .4
```



Comments

1. توضیحات بخشی مشترک در همه زبان ها

```
/* Comment here */.2
```

```
// Comment here.3
```